ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería de Computadores

Lenguaje visual de programación por bloques para la construcción de ciudades inteligentes

A visual language for brick-based construction of smart cities

Realizado por

Miguel Jesús Lara Bermúdez

Tutorizado por

Eduardo Guzmán de los Riscos

Amalia Cristina Urdiales García

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio 2017

Fecha defensa:

El Secretario del Tribunal

Resumen:

El objetivo de este trabajo es realizar una extensión a un lenguaje de programación por bloques ya existente, dando la posibilidad de hacerlo compatible con algunas placas de desarrollo concretas, con la finalidad de poder programarlas con una mayor facilidad y de una forma visual, integrar dichas placas de desarrollo en un entorno real, como, por ejemplo, la construcción de ciudades inteligentes, y, por último, ayudar a aumenta la integración de la programación en la educación entre niños de ocho y diez años.

El cumplimiento de estos objetivos se ha llevado a cabo con una metodología de desarrollo software iterativa y creciente o incremental, iniciada con un análisis y finalizada con la instauración y aprobación del sistema. La extensión se ha llevado a cabo sobre el proyecto educativo y de código abierto Bitbloq, desarrollado por la empresa BQ, el cual nos ha proporcionado las características necesarias para poder abarcar los objetivos finales del proyecto.

Palabras claves: lenguaje visual de programación, placas de desarrollo, ciudades inteligentes, educación.

Abstract:

The purpose of this project is to make an extension to a current brick-based visual programming language, offering the possibility of compatibility with some specific development boards, so that you can program them visually and in an easier way, as well as integrating these development boards in a real environment, for example, building smart cities. Finally, this helps increasing programming on education among children aged between eight and ten.

In order to fulfill these purposes, we have chosen an interactive and growing or incremental software development methodology. This methodology starts with an analysis and finishes with the acceptance and approval of the system. The extension has been carried out on the educational and open-source project Bitbloq, it is developed by the company BQ. We have found necessary features in this project to be able to get the final objectives of this project.

Keywords: programming visual language, development boards, smart cities, education.

Índice

1. Introducción

La evolución de la tecnología en nuestra vida cotidiana y la introducción de las Tecnologías de la Información y la Comunicación en el ámbito de la educación, han hecho que surjan nuevos usos de estas que muchos están empezando a destacar, como la introducción de la programación como herramienta educativa. Los estudios sitúan que, en España, la adopción de la programación en la enseñanza es escasa en la mayoría de los centros escolares del país, aunque existen varias campañas para promover la programación en la educación.

Una de los principales motivos de apostar por la programación a nivel educativo es para que los estudiantes no sean solamente consumidores de contenido, sino que también tengan la oportunidad de crearlo y sean autosuficiente para afrontar los problemas que puedan surgir. Una serie de valores que aporta la programación y que otras disciplinas no tienen, es la de ser estructurados, metódicos y organizados. Varios estudios demuestran que los niños que aprender a programar obtienen mejores resultados en pruebas de matemáticas, de razonamiento y de resolución de problemas. Ayuda a aumentar la capacidad de atención, más autonomía y consiguen mayor grado para habilidades cognitivas y socio-emocionales.

Existen numero grupos de investigación y universidades que han desarrollado numerosas herramientas que han tratado de facilitar la enseñanza y el aprendizaje de la programación, para que el alumnado programe de una forma más sencilla, sin tener que entrar en detalles de sintaxis ni de errores de compilación. Por ejemplo, el trabajo desarrollado por el grupo Lifelong Kindergarten en el Media Laboratory del MIT destaca por encima del resto, desarrollando herramientas como Scratch y App Inventor, que cuentan con cientos de miles de usuarios en todo el mundo.

El objetivo inicial de este trabajo fin de grado fue desarrollar una capa de abstracción del mismo nivel que ofrece Scratch, pero implementada en la plataforma de Bitbloq, a través de la cual, se pudiese programar microcontroladores y/o microprocesadores de una forma más sencilla y ofreciendo una capa visual cercana al ser humano y

orientada al aprendizaje del Internet de las cosas entre los niños, sin depender del tipo de microcontrolador o microprocesador que se tenga en la capa del hardware.

Este objetivo inicial se vio afectado en el inicio de su desarrollo, ya que se encontró una ineficiencia que se debía resolver para poder continuar con dicho objetivo. Por lo que, el objetivo que a continuación se exponen se convirtió en el principal de este trabajo fin de grado.

Este objetivo consiste en desarrollar una extensión a la base que nos ofrece la plataforma de Bitbloq. A través de esta extensión se ofrecerá un método más optimizado para realizar las conexiones entre los componentes y las placas de desarrollo que soporta dicha plataforma.

Este proyecto tiene un carácter eminentemente educativo, ya que está orientado a la enseñanza de la programación para alumnos de Primaria, y a cómo podrían, de una forma gráfica e intuitiva, programar un sistema con sensores y actuadores para, por ejemplo, construir una ciudad inteligente con piezas de Lego. Será, por tanto, de especial relevancia que la interfaz está diseñada para este perfil de usuarios.

Esta memoria queda estructura en las partes que a continuación presentamos:
En el siguiente punto, tecnologías y herramientas utilizadas, se comentará las tecnologías que han sido necesarias utilizar en cada una de las partes que conforman el proyecto, así como las distintas herramientas que se han utilizado para facilitar el desarrollo de dicho proyecto.
En el punto tres, estudio de interfaces de programación de aplicaciones, se presentarán y se llevará a cabo un estudio de dos plataformas que nos permiten utilizar programación por bloques, se expondrá también los pros y los contras que hemos considerado, así como las conclusiones y la elección final de la plataforma desde la cual partirá el proyecto.
En el punto cuatro del documento, se desarrollará una explicación de los pasos que hemos realizado para llevar a cabo la extensión del proyecto Bitbloq, así como también se explicará los ficheros que hemos necesitado modificar para ello.

En el quinto punto, se llevará a cabo una descripción de las distintas pruebas que hemos ido realizando, y el sexto punto del documento, se comentarán los resultados obtenidos de dichas pruebas. Por último, se añadirá los anexos I y II, donde se desarrollará el manual de usuario y el manual de instalación, respectivamente. Para finalizar, se desarrollará un apartado donde se expondrá todas las conclusiones obtenidas del proyecto.

2. Tecnologías y herramientas utilizadas

A continuación, se va a proceder a citar las distintas tecnologías que se han necesitado para llevar a cabo el proyecto. Este proyecto, se divide en tres partes fundamentales, que son: el frontend, el backend y la aplicación web2board. Todas las partes con distintas tecnologías de desarrollo. Añadir que este proyecto solamente es compatible con el navegador Google Chrome, ya que utiliza el motor de JavaScript V8 de dicho navegador.

El frontend, es la parte que interactúa con el usuario y es ejecutado por el navegador. Esta parte, está desarrollada con la tecnología de Angular JS (1.x), framework de JavaScript, creado y mantenido por Google y de código abierto. Sigue el Modelo Vista Controlador (MVC) como modelo de programación, y tiene como objetivo aumentar las aplicaciones ejecutadas en el navegador.

La parte del backend, es la parte que se ejecuta en el servidor e interactúa con la base de datos. En esta ocasión, está desarrollado en Node.js a través del framework Express. NodeJS es un entorno de ejecución, de código libre, para JavaScript y construido con el motor de JavaScript V8 de Google Chrome. Fue desarrollado con el objetivo de crear aplicaciones web altamente escalables. Utiliza un modelo de operaciones de entrada y salida (E/S) sin bloqueo y orientada a eventos, lo cual lo hace liviano y eficiente.
El backend es el encargado de proporcionar la librería de bloques; esta plataforma hace uso de la librería llamada Bloqs, una librería de bloques visuales de programación, desarrollada con JavaScript y creada para hacer más fácil la programación de robótica.

La aplicación web2board, es la aplicación que se utiliza para que la aplicación web sea capaz de reconocer la placa de desarrollo conectada al ordenador del usuario, es decir, es el punto de conexión entre la aplicación web y la placa que se utilice en el desarrollo. Esta aplicación, está desarrollada en Python, lenguaje de programación interpretado multiparadigma, ya que es capaz de soportar orientación a objetos, programación imperativa y programación funcional, en menor medida. Este lenguaje

se desarrolló con el objetivo de tener una sintaxis más clara y de mayor facilidad para su lectura.
Web2board hace uso del ecosistema de código libre “PlatformIO”, el cual proporciona cientos de librerías para las placas más populares de desarrollo de IoT.

El proyecto, no cuenta solamente con las partes que he nombrado anteriormente, también es necesario una base de datos, la cual es accedida por el backend. En este caso, es necesaria la utilización de MongoDB, un sistema de bases de datos NoSQL, orientado a documentos y de código abierto.

Aparte de las tecnologías nombradas anteriormente, necesarias para poder llevar a cabo el funcionamiento de la plataforma, se han utilizado tecnologías que nos han ayudado a crear el entorno de desarrollo, entre ellas se encuentran:

- Grunt: librería de JavaScript que nos permite realizar la configuración para automatizar tareas
- Bower: gestor de paquetes para HTML, CSS, JavaScript, fuentes y ficheros de imágenes
- NPM: gestor de paquetes de JavaScript, el cual viene agregado a Node.js

Además de estas tecnologías, hemos hecho uso de herramientas para el desarrollo del proyecto, entre ellas están:

- WebStorm: entorno de desarrollo integrado (IDE) de JavaScript
- Studio 3T: herramienta que nos permite la gestión de MongoDB
- Insomnia: herramienta utilizada para depurar API REST

3. Estudio de plataformas para programación por bloques

En este punto del proyecto, se ha desarrollado el estudio de dos plataformas para la programación por bloques, dichas plataforma son Bitbloq y Scratch. El motivo por el cual solamente se ha elegido estas dos plataformas para realizar el estudio ha sido: en primer lugar, porque las dos plataformas son de código abierto, y, en segundo lugar, porque son las únicas plataformas que no son extensiones de otras, ya que las demás plataformas existentes hasta el día de hoy, como, por ejemplo, ScratchX, mBlock, Snap… son extensiones de Scratch.

Tanto Bitbloq como Scratch, usan el mismo paradigma de programación: hacer uso de bloques para construir programas que encajan como las piezas de un puzzle. Esto hace, que la programación sea accesible a los niños, ya que no se tiene que preocupar de la sintaxis del código.

Bitbloq es una plataforma de código abierto y de uso gratuito, creada en el año 2014 para la programación por bloques de placas controladoras y robots, y desarrollada por BQ. Este proyecto se inició con la idea de ser utilizado en las enseñanzas relacionadas con la ciencia, tecnología, ingeniería, arte y matemáticas, conocidas como las enseñanzas STEAM, y desvincular la electrónica y la robótica como conceptos complicados. Desde hace varios años, Bitbloq ha sido una de la herramienta más utilizada para la introducción a los niños en la programación de robots; esta plataforma no solo te permite desarrollar sencillos proyecto, sino que también tiene la capacidad de poder albergar proyecto complejos.

En la última versión de la plataforma, Bitbloq 2, se realizó un cambio en la interfaz, siendo ésta más amigable que en la versión anterior; se añadió su propio IDE de Arduino, por lo que te permite programar tus placas controladoras usando tanto bloques como código; se dio la posibilidad de añadir las librerías necesarias para la placa de desarrollo que se esté utilizando; también se realizó una extensión de los bloques, para realizar funciones más complejas. Estos fueron los cambios más importantes que hubo respecto a la versión anterior.

A día de hoy, este proyecto soporta las siguientes placas de desarrollo y robots de forma estable:

Placa/Robot	Fabricante
Bq ZUM	BQ
Freaduino Uno	Freaduino
Arduino Uno	Arduino
Arduino Mega 2560	Arduino
Zowi	BQ
Evolution	BQ
mRanger	Makeblock
mBot	Makeblock

Tabla 1. Placas y robots compatibles con Bitbloq

Y se encuentran en desarrollo los microcontroladores Arduino Nano y Leonardo.

Pros	Contras
Desarrollado con orientación a la programación de placas controladoras	Solamente funciona en el navegador Google Chrome
Versión online y offline	La interfaz gráfica no se adapta a todo tipo de pantallas
Compatible con todos los sistemas operativos (Windows, Linux, Mac OS)	Si eliges la versión offline, no obtendrás las últimas actualizaciones, no tendrás una sincronización de los proyectos con tu cuenta de Bitbloq, no podrás hacer modificaciones en el código que genera el proyecto y no podrás compartir tus proyectos con otros usuarios
Solamente necesitas instalar la aplicación Web2board	Solamente se puede utilizar con las placas de Arduino y BQ, aunque se está desarrollando la compatibilidad con otros fabricantes
Se puede hacer un uso completo de la plataforma de forma gratuita	En algunos momentos del desarrollo, no se ha encontrado la documentación que se necesitaba
Interfaz de usuario intuitiva	Orientado, solamente, a la programación para el Internet de las cosas

Gran soporte por parte de la comunidad, tanto para los desarrolladores como para los usuarios	
Proyecto activo en cuanto a desarrollo	
Permite la compartición de proyecto con otros usuarios	
Nos permite programa las placas de desarrollo con una gran facilidad	

Tabla 2. Ventajas y desventajas encontradas en Bitbloq

Scratch es una de código abierto y de uso gratuito, al igual que Bitbloq, que empezó a desarrollarse en el 2002 por MIT Media Lab. Está orientado a la programación de juegos, historias animadas o mini videojuegos. También puede usarse para un gran número de propósitos educativos, como proyecto de ciencias o matemáticas, incluyendo simulación y visualización de experimentos. Supuso un avance en la compresión de la efectividad y el diseño innovador de las tecnologías de la información y la comunicación (TIC) para mejorar el aprendizaje en los centros escolares. Esta plataforma no es únicamente utilizada en los centros educativos, el uso de la plataforma llega hasta los museos, bibliotecas, centro comunitarios y hogares.

Existen varios lenguajes por bloques basados en Scratch, los cuales añaden algunas extensiones al lenguaje original, como cambios en la interfaz gráfica del usuario, la sintetización de sonido, la capacidad de programar microcontroladores de Arduino, la reproducción de mensajes de texto, entre otras.

En la última versión de esta plataforma, Scratch 3.0, traerá consigo nuevos tipos de bloques: bloques horizontales, para los usuarios principiantes, y bloques verticales, para los usuarios más avanzados. Estos nuevos bloques están basados en la librería de bloques desarrollada por Google, Blocky. También se hará un nuevo diseño más adaptable a las pantallas de los teléfonos y tabletas. Se podría decir que esta nueva versión es la divergencia entre el Scratch original y la versión para tabletas, Scratch Jr. Añadir, que esta versión sigue todavía en desarrollo, por lo que es posible, que se añadan algunas mejoras más respecto a las que se han referenciado.

Pros	Contras
Se puede hacer un uso completo de la plataforma de forma gratuita	La interfaz gráfica del entorno offline no es una interfaz amigable, además de ser una versión poco estable, incluso se recomienda utilizar la versión online
Versión online y offline	Necesitas usar algunas de las plataformas extendidas si quieres programar placas de desarrollo
Compatible con todos los sistemas operativos (Windows, Linux, Mac OS)	El programa offline solamente es compatible con los sistemas operativos de 32 bits
Permite la compartición de proyecto con otros usuarios	Proyecto poco activo en cuanto a desarrollo, Scratch 3.0, lleva desde el año 2015 en desarrollo
Gran soporte, tanto para los desarrolladores como para los usuarios	
Existe muchas plataformas basadas en Scratch	
Nos permite crear animaciones y juegos, de forma que podemos programar el comportamiento de objetos como por ejemplo un animal	

Tabla 3. Ventajas y desventajas encontradas en Scratch

Tras hacer una breve descripción y exponer los pros y los contras que hemos encontrado de las dos plataformas que hemos visto más adecuadas para las necesidades que se nos presenta en este proyecto, hemos decidido elegir la plataforma Bitbloq para llevar a cabo dicho proyecto.

Las razones por las cuales se ha realizado la elección de esta plataforma son: que desde un principio esta plataforma se desarrolló para la programación de placas de desarrollo, y no con otra finalidad, como fue la idea de Scratch y que más tarde, concretamente en la versión 2.0, se añadió la extensión de poder llevar a cabo la programación de placas de desarrollo. Otro motivo por el cual hemos realizado la elección, ha sido el uso de PlatformIO en esta plataforma, ya que da soporta a una gran cantidad de placas de desarrollo, y entre ellas se encuentra la placa que desde

un primer momento se iba a utilizar para este proyecto. Aparte, la plataforma es totalmente online, simplemente se tiene que realizar la instalación de la aplicación web2board.

A pesar de las razones expuestas anteriormente, Bitbloq no nos proporcionaba el nivel de abstracción que nos ofrecía Scratch, que era el nivel que buscábamos, pero si nos ofrecía la compatibilidad con el hardware que desde un primer momento se pensó utilizar, por lo cual decidimos seguir con este proyecto como base y realizar el desarrollo de la optimización que vimos necesaria para futuras extensiones, y que no nos proporcionaba Bitbloq.

4. Desarrollo de optimizaciones

Tras un tiempo analizando la organización de la plataforma de Bitbloq y realizando algunas pruebas para ver el comportamiento de la misma, nos fijamos en que el sistema le permite al usuario conectar los componentes a cualquier pin de la placa que el usuario seleccione, siempre que el pin de la placa sea del mismo tipo que el pin del componente, es decir, que si el pin del componente es de tipo digital pues el usuario será libre de elegir cualquier pin digital de la placa que haya seleccionado.

Este comportamiento no tiene ninguna repercusión mientras estamos conectando componentes dentro de la plataforma, pero sí la tiene cuando queremos realizar esas conexiones físicamente y más aún cuando pensamos en que el espacio disponible para colocar la placa de desarrollo y los componentes puede ser limitado. Además, pensando en que el objetivo final de este proyecto es que los niños no tengan que adquirir ningún conocimiento del hardware que se esté utilizando.

Partiendo de esta idea, las optimizaciones que deberíamos realizar serían: por una parte, intentar que las conexiones estén lo más próximas físicamente posible, por ejemplo, si un sensor tiene dos terminales de datos, no usar un GPIO0 y un GPIO10, ya que en el momento de realizar la conexión física se complica la conexión, y, por otro lado, intentar escoger los terminales que menos multiplexados estén, o los que estén multiplexados con elementos que no se vayan a usar más adelante, por ejemplo, mejor poner un botón en un GPIO que funcione como entrada de un reloj externo de comunicaciones, que seguro va a tener menos uso, que de uno que genere PWM para activar motores.

Las necesidades que nos surgieron se implementarían en la parte del frontend de la aplicación, ya que es la parte encargada de gestionar las interacciones del usuario con la aplicación, y, por tanto, la parte que lleva a cabo las conexiones entre componentes y placas.

La aplicación hace uso de una librería en JavaScript llamada “jsplumb”. Esta librería proporciona la conexión visual entre elementos de las páginas web, en este caso, los

elementos serían los componentes y la placa seleccionada por el usuario. Existe también un fichero que hace uso de dicha librería, y en el cual se encuentran las funciones a través de las cuales se llevan a cabo el control de las desconexiones entre componentes y placa, el control de añadir o borrar componentes y placas, también se realiza el autoguardado del proyecto del usuario, así como cualquier interacción que el usuario pueda hacer con dichos elementos. Dicho fichero es nombrado como *hw2bloqs.js* y se encuentra en el directorio *./app/scripts/services* del proyecto.

Teniendo en cuenta los objetivos que se comentaron anteriormente, se decidió que la mejor forma para llevarlos a cabo sería extender las funciones desarrolladas en este fichero, exactamente, la extensión se desarrollaría sobre la función que ejecuta la acción de guardar el proyecto que el usuario haya creado, de forma que cada vez que se guardase alguna modificación, se realizará la optimización en las conexiones de los componentes.

A continuación, se explicará el código desarrollado, a través del cual se cumpliría las optimizaciones descritas anteriormente.

```
exports.saveSchema = function() {

    var schema = {
        components: [],
        connections: []
    };
    var endpointsRef = {};

    function _setParameters(ep) {
        endpointsRef[ep.getParameter('pinComponent')] = {
            uid: ep.getUuid(),
            type: ep.scope
        };
    }

    var componentList = [].slice.call(containerDefault.querySelectorAll('.component'));
    componentList.forEach(function(item) {

        var endpoints = jsPlumbInstance.getEndpoints(item);

        if (endpoints && endpoints.length > 0) {

            endpointsRef = {};
            endpoints.forEach(_setParameters);

            var connections = jsPlumbInstance.getConnections({
                source: item
            });

            if (connections.length) { //components disconnected are not saving
                schema.components.push({
                    endpoints: endpointsRef,
                    uid: item.dataset.uid,
                    connected: connections.length > 0
                });
            }
        }
    });

//Store connections data
    schema.connections = _getConnections();

    return schema;
};
```

Figura 1. Función saveSchema

Esta función es la encargada de obtener y devolver el esquema del proyecto que el usuario ha creado y el cual será guardado en la base de datos para más tarde obtenerlo y cargar de nuevo el proyecto tal y como se guardó, es decir, en esta función es donde se obtiene la placa, los componentes y las conexiones utilizadas en el proyecto. Es en este código, donde se ha introducido el código desarrollado.

Antes de pasar a la explicación del código desarrollado, añadir que se ha necesitado la declaración de una estructura de datos, concretamente una matriz unidimensional, que ha sido nombrada como “arrayComponentes”, en la cual se añade el código HTML de los componentes que el usuario vaya eligiendo, esto nos permitirá, como explicaremos más adelante, obtener los pines del componente. También se ha necesitado declarar de forma general otra variable que hemos nombrado “nuevasConexiones” y en la cual se almacena las nuevas conexiones que nuestro código ha realizado y que finalmente será las conexiones devueltas por la función anterior. También se utiliza otra estructura de datos, ya declarada en el código, para almacenar las conexiones anteriores entre los componentes y la placa.

Las dos últimas variables necesarias para nuestro código han sido, por un lado “componentesConPinesAsignados”, en esta estructura de datos de tipo matriz unidimensional se almacenará los componentes cuyos pines están asignados, esto se hace para asegurar que componentes especiales vayan a los pines adecuados, nos referimos a componentes especiales a aquellos que el pin donde se vaya a conectar tenga unas características especiales, por ejemplo, un motor necesitará que el pin de la placa donde se vaya a conectar sea capaz de producir una onda PWM. La última estructura de datos, es la variable nombrada como “componentesSinConexion”, en la cual se almacenará todos los componentes que el usuario agregue; en este caso se almacena una información distinta a la que se almacena en la variable “arrayComponentes”, esto nos ayudará a saber si un componente que estaba conectado ha sido desconectado, si es así, dicho componente deberá ser conectado de nuevo. Esta situación se puede producir cuando un componente está conectado a un pin de la placa y dicho pin sea necesario para algunos de los componentes con pines asignados que hemos explicado anteriormente.

Una vez explicado las distintas variables de tipo generales declaradas en el código vamos a realizar la explicación de las funciones que se han desarrollado.

```
function _obtenerNombre(uid) {
      var nombre = '';
      var pines = board.pins;

      for (var type in pines) {
          if (pines[type]) {
              pines[type].forEach(function (pin) {
                  if (uid === pin.uid){
                      nombre = pin.name;
                  }
              });
          }
      }
      return nombre;
  }
```

Figura 2. Función obtenerNombre()

La función que aparece en la imagen anterior, figura 2, es utilizada para obtener el nombre de un pin de la placa de desarrollo utilizada. Para ello, pasamos como parámetro el identificador del pin del cual se desea obtener su nombre y vamos iterando sobre todos los pines de la placa, finalmente se devolverá el nombre del pin, en el caso de que no se encuentre dicho pin se devolverá una cadena vacía.

```
function _estaEnUso(uid){
      var enUso = false;
      nuevasConexiones.forEach(function (conexion) {
          if (conexion.pinTargetUid === uid){
              enUso = true;
          }
      });
      return enUso;
  }
```

Figura 3. Función estaEnUso

Esta función se encarga de devolver una variable de tipo booleano cuyo valor depende de si un pin de la placa se está utilizando o no en las conexiones que nuestro algoritmo está realizando. Para ello, se pasa como parámetro el identificador del pin y se compara con el pin destino (*conexion.pinTargetUid*) de cada nueva conexión almacena en la matriz unidimensional “nuevasConexiones”.

```
function _obtenerNuevoPinDestino(tipo) {
      var pinResultado = null;
      for (var i in board.pins[tipo]){
          var pin = board.pins[tipo][i];
          if (!_estaEnUso(pin.uid)){
              pinResultado = pin;
              break;
          }
      }
      return pinResultado;
 }
```

Figura 4. Función obtenerNuevoPinDestino

A través de esta función se obtiene el nuevo pin perteneciente a la placa, al cual se conectará el componente. Para ello, vamos recorriendo todos los pines de la placa que sea del mismo tipo (digital, analógico…) que el pin del componente, el tipo del componente es pasado como parámetro a la función, y se devolverá el primer pin que encontramos que no esté en uso, para ello, como se puede observar en el código, hacemos uso de la función explicada anteriormente.

```
function _buscarConexion(origen) {
    var cnx = null;
    schema.connections.forEach(function (conexion) {
       if (conexion.pinSourceUid === origen){
           cnx =  conexion;
       }
    });
    return cnx;
}
```

Figura 5. Función buscarConexion()

La función que se muestra en la figura 5, devolverá la conexión que tiene como pin origen el pin que es pasado como parámetro, concretamente, el parámetro de la función hace referencia al identificador de un pin perteneciente a un componente, en otro caso, se devolverá una variable con un valor nulo. Para ello, se iterará sobre las conexiones que estaban realizadas la última vez que se ejecutó el guardado del programa, y se comparará el identificador de cada pin origen correspondiente a las conexiones con el parámetro de la función.

```
function _obtenerPines(uid) {
    var pines = [];
    arrayComponentes.forEach(function (component) {
        if (component.uid === uid){
            pines =  component.pins;
        }
    });
    return pines;
}
```

Figura 6. Función obtenerPines

Esta función devuelve los pines de un componente cuyo identificador es parámetro de la función. Para ello, se recorre la matriz unidimensional donde se almacena los componentes que el usuario ha seleccionado y se compara el identificar de cada componente con el que es pasado como parámetro, cuando los identificadores coinciden, se obtiene los pines desde la matriz unidimensional y se devuelven. En el caso de que el componente no se encuentre se devuelve un vector sin datos.

```
function _obtenerEP(pin, component) {
    return component.endpoints[pin];
}
```

Figura 7. Función obtenerEP

La función que se muestra en la imagen es utilizada para obtener lo que es llamado como punto final del componente. Para ello pasamos como parámetros de la función el componente conectado por el usuario y un pin perteneciente al dicho componente. Los puntos finales son utilizados por la librería “jsPlumb” para indicar que con ese punto se puede realizar una conexión. De aquí podemos concluir que todos los pines de la placa y de los componentes serán tratados como puntos finales para poder llevar a cabo la conexión entre ellos.

```
function _estabaConectado(uid) {
      var conectado = false;
      if (oldConnections.length){
          oldConnections.forEach(function (conexion) {
              if (conexion.pinSourceUid === uid){
                  conectado = true;
              }
          });
      }
      return conectado;
  }
```

Figura 8. Función estabaConectado()

Esta función es utilizada para chequear si algunos de los componentes de los que no están conectados estaba anteriormente conectados. Con anteriormente, nos referimos a la última vez que se realizó un guardado del proyecto. Para ello utilizamos una estructura de datos donde se almacena las ultimas conexiones que se realizaron y se itera sobre dicha estructura. Si el identificador que es pasado como parámetro y que corresponde con el identificador del alguno/s de los pines del componente se encuentra en la estructura de datos significa que el componente tenía ese pin conectado, por lo que se devolverá una variable de tipo booleano indicando que estaba conectado. En otro caso, se indicará que no estaba conectado anteriormente.

```
function _tienePinAsignados(uid, p) {
    var tiene = false;
    componentesConPinesAsignados.forEach(function (component) {
        if ((component.uid === uid) && component.pin[p]){
            tiene = true;
        }
    });
    return tiene;
}
```

Figura 9. Función tienePinAsignados()

Utilizando los parámetros de esta función, que se corresponde con el identificador de un componente y algunos de sus pines, se busca dicho identificador en la estructura donde se almacena los componentes añadidos que tienen pines asignados, y se comprueba si para el pin recibido existe un valor en el campo del componente que corresponde con la matriz unidimensional de pines asignados (*component.pin*). A

través de esta función comprobamos si el componente tiene pines asignados. Se utilizará para ejecutar o no, el método de obtener un nuevo pin destino, ya que, si el componente tiene pines asignados, no será necesario buscar unos nuevos pines para realizar la conexión con el componente.

```
function _obtenerPrioridad(componente) {
    var prioridad = 0;
    var pines = _obtenerPines(componente.uid);
    for (var type in pines) {
        if (pines[type]) {
            prioridad += pines[type].length;
            pines[type].forEach(function (pin) {
                if (_tienePinAsignado(componente.uid, pin)){
                    prioridad++;
                }
            });
        }
    }
    return prioridad;
}
```

Figura 10. Función obtenerPrioridad()

La figura anterior representa la función que es utilizada para obtener la prioridad de un componente, esta prioridad será utilizada para ordenar los componentes que son agregados y se dará más prioridad a los componentes con mayor número de pines y con mayor número de pines asignados. De esta forma se buscará una conexión válida a un componente que tenga dos pines antes que a uno que tenga un pin solamente, esta prioridad se verá aumentada si el componente ya tiene pines asignados.

Para calcular esta prioridad, el componente se recibe como parámetro de la función y se hace uso de funciones explicadas anteriormente para obtener sus pines, el número de pines será sumado a la prioridad del componente. Se iterará sobre los pines obtenidos para chequear si ese pin está ya asignado, si es el caso, la prioridad se verá aumentada en uno por cada pin asignado.

```
function compare(a, b) {
    var pa = _obtenerPrioridad(a);
    var pb = _obtenerPrioridad(b);

    if (pa < pb){
        return 1;
    }
    if (pa > pb){
        return -1;
    }

    return 0;
}
```

Figura 11. Función compare()

Esta función será utilizada por el método de ordenamiento de JavaScript para asignar un índice dentro del vector de objetos según el resultado devuelto por esta función, en nuestro caso, el índice se asignará según la prioridad de los componentes. Por tanto, se obtendrá las prioridades de los dos objetos que son parámetros de la función, y se procederá como sigue:

- Si la prioridad del componente “a” es menor que la prioridad del componente “b”, se devolverá un 1, indicando que el componente “b” deberá obtener un índice menor que el del componente “a”
- Si la prioridad del componente “a” es mayor que la prioridad del componente “b”, se devolverá un -1, indicando que el componente “a” deberá obtener un índice menor que el del componente “b”
- Si las prioridades son iguales, se devuelve un 0, y no se altera el orden entre “a” y “b”, pero ordenados con respecto a los demás componentes

Ha sido necesario la declaración de esta función porque por defecto el método de ordenamiento de JavaScript actúa de forma inversa a la descrita en esta función.

```
function _reordenarConexiones() {
    schema.components.forEach(function (component) {
        if (component.connected){
            var pines = _obtenerPines(component.uid);
            for (var type in pines) {
                if (pines[type]) {
                    pines[type].forEach(function (pin) {
                        var ep = _obtenerEP(pin, component);
                        var tipo = ep.type;
                        var pinOrigen = ep.uid;
                        var nuevoPinDestino = _obtenerNuevoPinDestino(tipo);
                        if (nuevoPinDestino){
                            var nuevaConexion = {
                                pinSourceUid: pinOrigen,
                                pinTargetUid: nuevoPinDestino.uid
                            };
                            nuevasConexiones.push(nuevaConexion);
                        }
                    });
                }
            }
        }
    });
}
```

Figura 12. Función reordenarPines()

La función que corresponde con la figura 12, es la encargada de reordenar las conexiones que el usuario realice; para ello hace uso, como podemos observar, de las distintas funciones explicadas anteriormente.

De forma general, esta función es la encargada de buscar un nuevo pin entre los pines disponibles de la placa y para cada pin de cada componente que ya está conectado a la placa seleccionada. La nueva conexión que se realice será almacenada en la variable “nuevasConexiones”.

```
function _conectar() {
    exports.disconnectAllComponents();
    nuevasConexiones.forEach(function (conexion) {
        if(conexion){
            jsPlumbInstance.connect({
                uuids: [conexion.pinSourceUid, conexion.pinTargetUid],
                type: 'automatic'
            });
        }
    });
}
```

Figura 13. Función conectar()

Esta función, se encarga de desconectar todos los componentes que hasta el momento están conectados, para ello, se hará uso de una función ya desarrollada, *disconnectAllComponents()*. También se encargará de realizar la conexión visual de las nuevas conexiones que el programa realice; utilizando una de las funcionalidades que tiene disponible la librería "jsPlumb".

Cuando nos referimos a conexión visual, nos estamos refiriendo a lo mostrado en la figura 14.

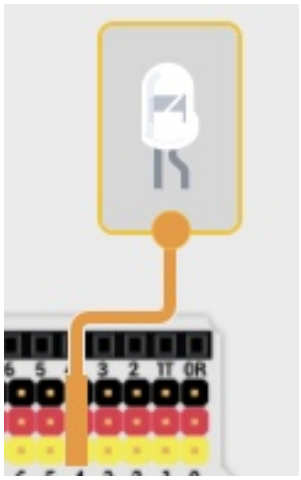

Figura 14. Conexión visual

```
function _chequearComponentesSinConexiones() {
    componentesSinConexiones.forEach(function (component) {
        var pines = _obtenerPines(component.uid);
        for (var type in pines) {
            if (pines[type]) {
                pines[type].forEach(function (pin) {
                    var ep = _obtenerEP(pin, component);
                    var tipo = ep.type;
                    var pinOrigen = ep.uid;
                    if (_estabaConectado(pinOrigen)){
                        var nuevoPinDestino = _obtenerNuevoPinDestino(tipo)
                        if (nuevoPinDestino){
                            var nuevaConexion = {
                                pinSourceUid: pinOrigen,
                                pinTargetUid: nuevoPinDestino.uid
                            };
                            nuevasConexiones.push(nuevaConexion);
                        }
                    }
                });
            }
        }
    });
}
```

Figura 15. Función chequearComponentesSinConexion()

En esta función se realiza el procedimiento de buscar un nuevo pin perteneciente a la placa de desarrollo. Para cada pin de cada componente que el usuario añadió y el cual no tiene conexión. Para ello, se itera sobre la estructura de almacenamiento en la que se almacena los componentes sin conexión y se comprueba para cada componente si estaba conectado o no, en el caso de que sí estuviese conectado se procederá como en la función anterior, figura 12.

A través de esta función, se evitará que un componente que estaba conectado quede desconectado a consecuencia de que otro con pines asignados se haya añadido, y que entre los pines asignados de dicho componente se encontrase algunos de los pines utilizados en la conexión del primer componente.

Además de la funcionalidad de reordenar las conexiones, para obtener una mayor eficiencia de cara a las conexiones físicas, hemos añadido otras funcionalidades, como por ejemplo la posibilidad de descargar un archivo con formato JSON, cuyo contenido se corresponde a las conexiones tras ejecutar el algoritmo y proporcionará

la información necesaria para poder hacer las conexiones físicas entre el microcontrolador y los componentes. A continuación, se mostrará una imagen correspondiente al código desarrollado para esta funcionalidad.

```
function _descargarArchivo(contenidoEnBlob, nombreArchivo) {
    //creamos un FileReader para leer el Blob
    var reader = new FileReader();
    //Definimos la función que manejará el archivo
    //una vez haya terminado de leerlo
    reader.onload = function (event) {
        //Usaremos un link para iniciar la descarga
        var save = document.createElement('a');
        save.href = event.target.result;
        save.target = '_blank';
        save.download = nombreArchivo || 'conexion.json';
        var clicEvent = new MouseEvent('click', {
            'view': window,
            'bubbles': true,
            'cancelable': true
        });
        //Simulamos un clic del usuario
        save.dispatchEvent(clicEvent);
        //Y liberamos recursos...
        (window.URL || window.webkitURL).revokeObjectURL(save.href);
    };
    //Leemos el blob y esperamos a que dispare el evento "load"
    reader.readAsDataURL(contenidoEnBlob);
}
```

Figura 16. Función descargarArhivo()

Con la ejecución de la función que aparece en la figura 16, realizamos la descargar del archivo a través del navegador y de forma automática, dando la posibilidad al usuario de descarga el fichero o no. Para ello, crearemos un objeto de tipo FileReader, el cual nos permitirá leer el contenido del fichero de forma asíncrona. La lectura del fichero se realiza a través de la función *readAsDataUrl*, proporcionada por el objeto anteriormente creado. Tanto el contenido del fichero como el nombre, serán parámetros de la función.

Cuando se finalice la lectura del contenido, se procederá a la creación de una etiqueta HTML de tipo <a> utilizando la función *createElement()* que nos ofrece JavaScript. Este elemento crea un enlace a otras páginas de internet, archivos o ubicaciones dentro de la misma página. En nuestro caso, se creará el enlace al archivo creado y

añadiremos el atributo *download* para indicar al navegar web que descargue dicho archivo y dirija al usuario a la ventana que podemos observar en la figura 17. Añadir, que la acción de pinchar sobre el enlace se realizará a través de un evento de JavaScript, como podemos observar en la siguiente línea de código, *save.dispatchEvent(clicEvent).* Finalmente, liberaremos los recursos que la aplicación ha necesitado utilizar para llevar a cabo la acción.

La propiedad *onLoad* de los objetos de tipo FileReader contiene un controlador de eventos que se lanza cuando el contenido del fichero ha sido leído completamente, en nuestro caso, se lanzará la función que ejecuta lo explicado en el párrafo anterior.

En cuanto al contenido del fichero, pertenecerá a un objeto de tipo Blob, estos objetos te permiten crear fichero y tratarlos como objetos de JavaScript en el lado del cliente. Los Blobs representan datos que no necesariamente se encuentran en un formato nativo de JavaScript. Más información sobre este tipo de objeto de JavaScript se puede localizar en el enlace [4] de la bibliografía de este documento.

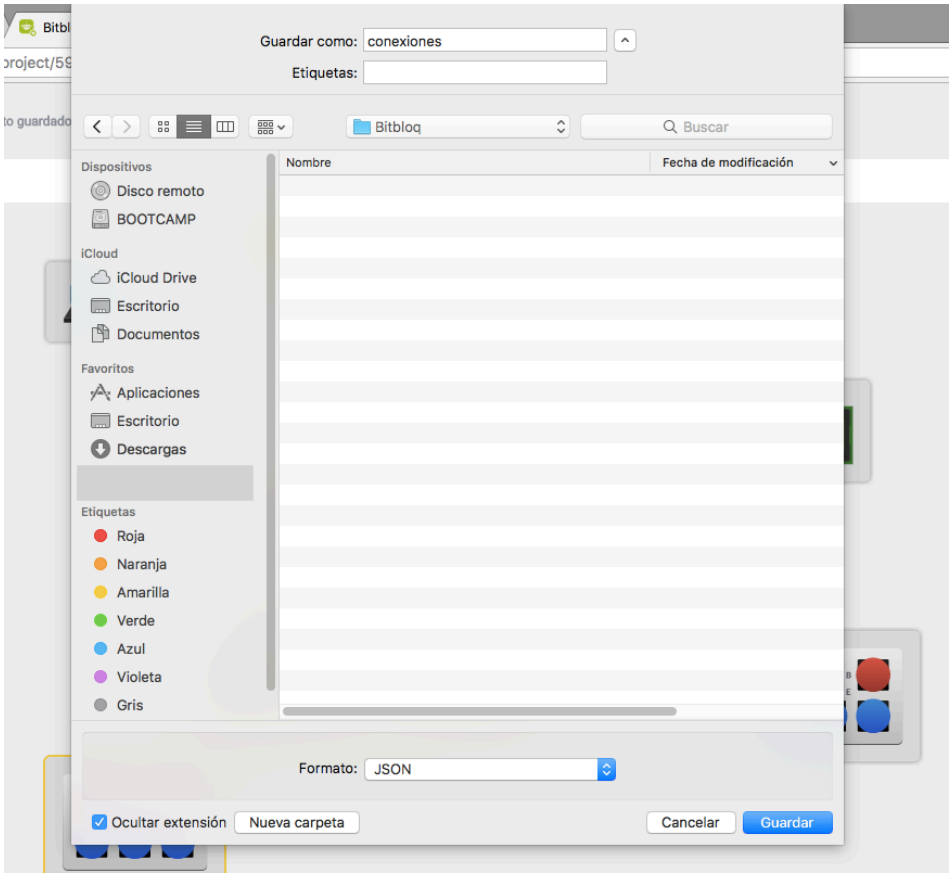

Figura 17. Ventana para guarda el archivo descargado

```
schema.connections = _getConnections();
schema.components.sort(compare);

_reordenarConexiones();
_chequearComponentesSinConexiones();
_conectar();

var fichero = JSON.stringify(nuevasConexiones);

if (window.confirm("¿Desea descargar el archivo con la conexiones?") === true) {
    _descargarArchivo(new Blob([fichero], {
        type: 'text/plain'
    }), 'conexiones.json');
}

localStorage.setItem('newconnections', fichero);

schema.connections = nuevasConexiones;
oldConnections = nuevasConexiones;
```

Figura 18. Llamadas a funciones

En el fragmento de código que muestra la figura anterior, podemos observar cómo se realizan las llamadas a algunas de las funciones explicadas anteriormente, así como la asignación de las nuevas conexiones a la variable que se utilizará para leer las conexiones en el momento de cargar el proyecto del usuario, corresponde con la variable “schema”. También encontramos la llamada a la función *sort* para la matriz unidimensional de componentes, a la cual se le pasa como parámetro la función de comparación que hemos desarrollado. Se puede consultar la documentación de esta función en el enlace [19] de la bibliografía.

Se observa también, la creación del objeto Blob, que es utilizado en la función *_descargarArchivo*, indicando en la creación el contenido y el nombre que se quiere utilizar para el fichero. El contenido del fichero, como se explicó anteriormente, se corresponde con las nuevas conexiones realizadas, pero realizando previamente una conversión de vector a cadena de tipo JSON, utilizando el método *JSON.stringify.* Haremos uso del objeto de JavaScript *Confirm Box* para que el usuario decida si desea o no descargar el archivo con las nuevas conexiones realizadas. En caso de aceptación, se procederá a ejecutar la función desarrollada para la descarga del archivo. La alerta que aparece se muestra en la figura 19.

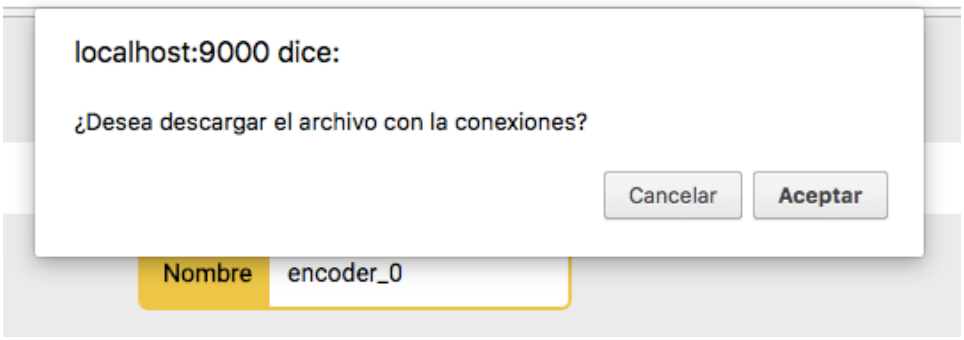

Figura 19. Mensaje de confirmación de la descarga

Las nuevas conexiones realizadas, no solo se permite almacenar en un fichero, sino que se almacena en el navegador, concretamente, se hace uso del *localStorage*, o como su propio nombre indica, se trata de un espacio de almacenamiento local, característica que se añadió con la aparición de HTML5. Algunas de las características de este almacenamiento, como se puede consultar en el enlace [7] de la bibliografía, son:

1. localStorage nos permite almacenar entre 5 y 10 megabytes de información, dependiendo del navegador.
2. La información almacenada con localStorage no será enviada en ningún momento al servidor.
3. Para que la información almacenada en el localStorage desaparezca, habrá que eliminarla expresamente, sino la información permanecerá almacenada, aunque se cierre el navegador.

Tras la explicación del código desarrollado para llevar a cabo las optimizaciones en las conexiones entre los componentes y la placa de desarrollo, vamos a realizar la explicación de algunas de las extensiones que hemos desarrollado para añadir más funcionalidad al proyecto de Bitbloq. Estas funcionalidades se añadirán al archivo perteneciente al proyecto *hardwareTab.js*, el cual se encuentra en el directorio: *./app./scripts/controllers*.

Estas funcionalidades añaden la opción de poder leer tanto las placas como los componentes desde un archivo con extensión JSON. Actualmente, la lectura de las características de los diferentes componente y placas se hace desde un archivo de JavaScript, donde se encuentran todas las posibles placas y componentes que

soporta el proyecto. Dicho archivo se debe mantener, ya que, si las funcionalidades que se explican a continuación fallasen, la información de las placas y de los componentes se obtendrá de este archivo, de esta forma, aseguramos que la aplicación siga funcionando correctamente. Con esta funcionalidad, será posible tener las características de las placas y de los componentes por separado, de forma que se podrá añadir más característica, o añadir nuevos componentes o placas de una forma más cómoda.

Estos ficheros se almacenarán en las siguientes rutas del proyecto:

- Placas: ./json/boards
- Componentes: ./json/components

```
var board;
dirJSON = './json/boards/' + data.id + '.json';
//Leer json
$http({
    method: 'GET',
    url: dirJSON
}).then(function successCallback(response) {
    board = response.data;
    _addBoard(board);
}, function errorCallback(response) {
    board = _.find($scope.hardware.boardList, function(board) {
        return board.id === data.id;
    });
    _addBoard(board);
});
```

Figura 20. Lectura de archivos JSON correspondientes a las placas

Para realizar la lectura del archivo JSON correspondiente a la placa que el usuario ha seleccionado, se lleva a cabo una petición a través del servicio $http de AngularJS, al cual se le indica el directorio donde se encuentra el archivo y el método de este servicio que queremos utilizar, en nuestro caso, GET. Aunque existen muchos módulos que hacen este trabajo de forma completa, es útil hacerlo sin depender de

ninguno, no siempre se va a utilizar módulos cuando creemos aplicaciones con AngularJS.

Como se observa en la figura 20, previamente hemos determinado la dirección donde se debería encontrar dicho archivo. Para ello hacemos uso del identificador de la placa, que es obtenido a través de la variable recibida como parámetro de la función de la figura 22, donde se encuentra el fragmento de código mostrado en la figura 20.

Si esta petición tiene éxito, se procederá a cargar la información contenida en el archivo a través de la función *_addBoard().* En otro caso, se ejecutará el código implementado originalmente del proyecto, el cual realiza una búsqueda de la placa seleccionada entre las distintas placas soportadas por el proyecto y devuelve la información (identificador de la placa, declaración de los tipos de pines que la placa contiene, declaración de los pines, etc.) correspondiente, la cual es la mínima para que la carga de la placa se realice con éxito, por lo cual, será la información mínima que el archivo JSON aportará.

```
var component;
dirJSON = './json/components/' + data.id + '.json';

//Leer json
$http({
    method: 'GET',
    url: dirJSON
}).then(function successCallback(response) {
    component = response.data;
    _addComponent(component);
}, function errorCallback(response) {
    _addComponent(data);
});
```

Figura 21. Lectura de archivos JSON correspondientes a los componentes

La lectura del fichero de tipo JSON correspondiente al componente seleccionado, se realiza de la misma forma que para la placa. Utilizando el servicio $http de AngularJS, se realiza una petición GET a la dirección que previamente definimos. Si esta petición tiene éxito, se pasará el contenido del archivo a la función *_addComponent()*; este

contenido especifica el identificador del componente, la declaración de los pines del componente, entre otros datos. En caso contrario, se pasará la variable que se recibe como parámetro de la función en la que el fragmento de código mostrado en la figura 21 está desarrollado, la cual se visualiza en la figura 22.

```
$scope.drop = function(data) {
    hw2Bloqs.userInteraction = true;
    switch (data.type) {
        case 'boards':
            $scope.currentProjectService.showActivation = false;
            $scope.currentProjectService.closeActivation = false;
            var board = _.find($scope.hardware.boardList, function(board) {
                return board.id === data.id;
            });
            _addBoard(board);
            $scope.changeToolbox('components');
            break;
        case 'components':
            if (!$scope.currentProject.hardware.board) {
                $scope.changeToolbox('boards');
                alertsService.add({
                    text: 'bloqs-project_alert_no-board',
                    id: 'error_noboard',
                    type: 'error'
                });
                return false;
            } else if ($scope.currentProject.hardware.robot) {
                alertsService.add({
                    text: 'bloqs-project_alert_only-robot',
                    id: 'error_noboard',
                    type: 'error'
                });
                return false;
            }
            _addComponent(data);
            break;
```

Figura 22. Fragmento de código original

La variable recibida como parámetro no contiene la misma información que nuestro archivo JSON, por lo que se tendrá que diferenciar si la información viene desde el archivo o desde la variable, esto se realiza en la función *_addComponent(),* mostrado a continuación en la figura 23.

Como hemos comentado anteriormente, la información varía según el origen de esta, por lo que solamente nos bastará comprobar si la información que recibimos está presente o no en ambos datos. Por ejemplo, en nuestro caso, sabemos que existe un

campo que solamente está presente en el archivo JSON, por lo tanto, comprobamos si existe ese campo; si es así, es que la información procede del archivo; en caso contrario debemos ejecutar el código que estaba desarrollado originalmente. En la figura 22, podemos observar lo explicado.

```
function _addComponent(data) {

    var component = data;
    if (!component.manufacter){
        component = $scope.componentsMap[data.id];
    }
    /*
    *
    * RESTO DEL CÓDIGO
    *
    */
}
```

Figura 23. Función addComponent()

5. Realización de pruebas

Conexión de dos componentes digitales

Esta prueba consiste en seleccionar una placa de desarrollo y dos componentes digitales entre los disponibles, los cuales conectaremos a los pines de la placa que correspondan. En nuestro caso, se ha realizado las conexiones que aparecen en la figura 24. Hemos seleccionado la placa de desarrollo BQ ZUM y como componentes digitales el LED y el botón.

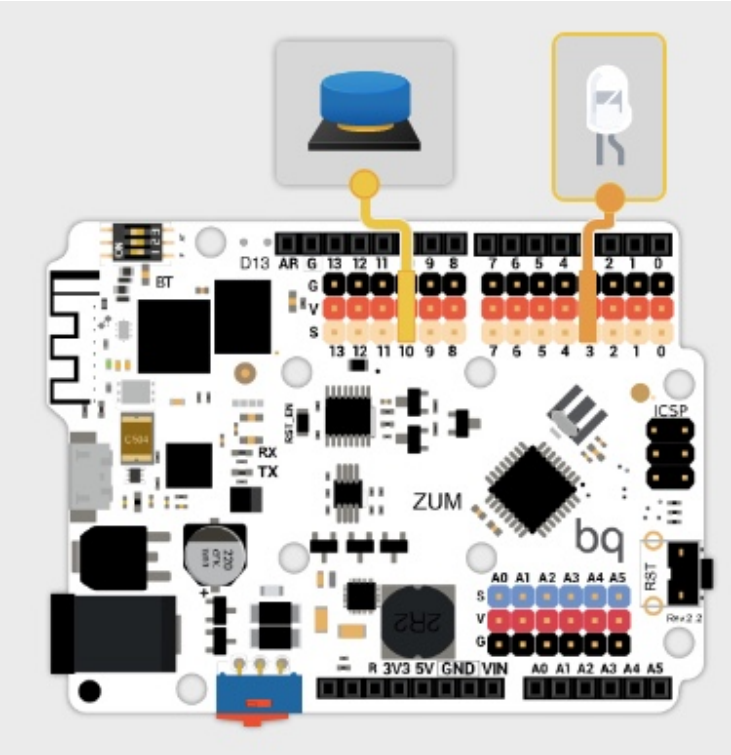

Figura 24. Conexión de componentes digitales

Los resultados que se obtendrán de esta prueba serán los mismos componentes pero conectados a pines consecutivos de la placa. Esto se debe a que el programa itera sobre los pines de la placa que son del mismo tipo que los componentes, en este caso digital, y devuelve el primer pin disponible. Como en esta ocasión solamente hay dos componentes, el programa devolverá los dos primeros pines consecutivos que no estén en uso. Estos resultados pueden ser observados en la figura 25, mostrada a continuación.

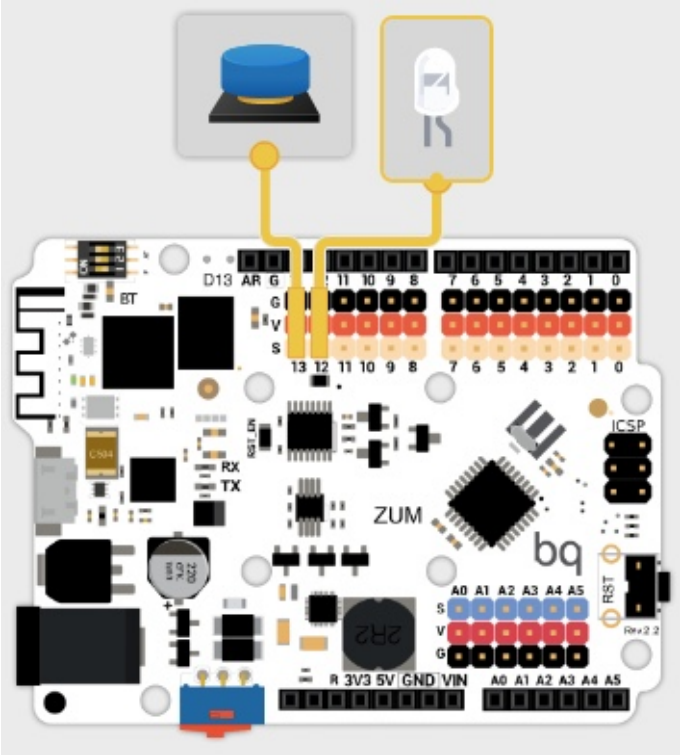

Figura 25. Resultado de conectar dos componentes digitales

Conexión de dos componentes analógicos

En esta otra prueba realizada, se ha seleccionado una placa de desarrollo y dos componentes analógicos, los cuales se han conectado a dos pines disponibles de la placa elegida y del mismo tipo que los componentes. En la figura 26, se muestra la placa y los componentes seleccionados, así como las dos conexiones que hemos decidido realizar. Se ha seleccionado la placa Freaduino UNO y como componentes analógicos el sensor de luz y la botonera.

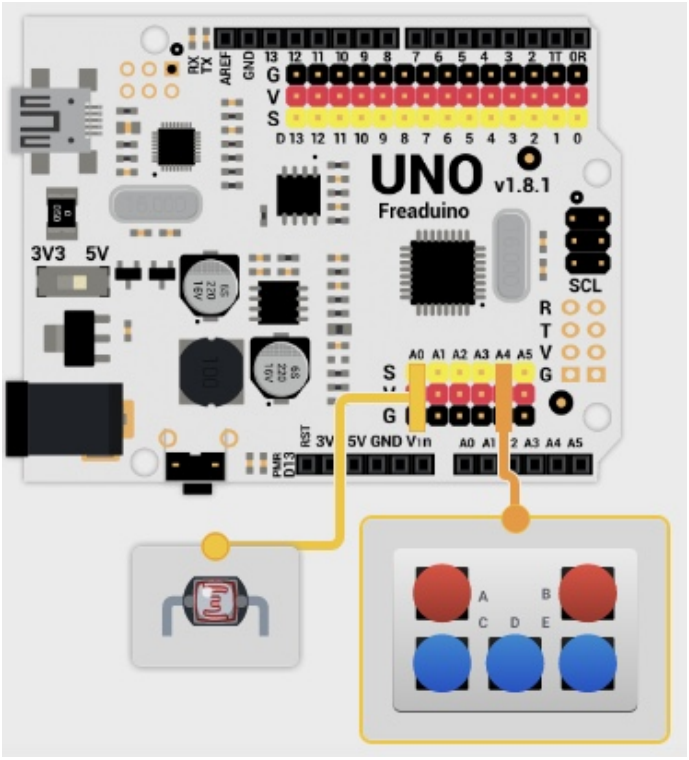

Figura 26. Conexión de componentes analógicos

Los resultados, como podemos observar en la figura 27, se corresponderán a los mismo que los de la prueba anterior, pero esta vez sobre los pines analógicos de la placa que hemos seleccionado.

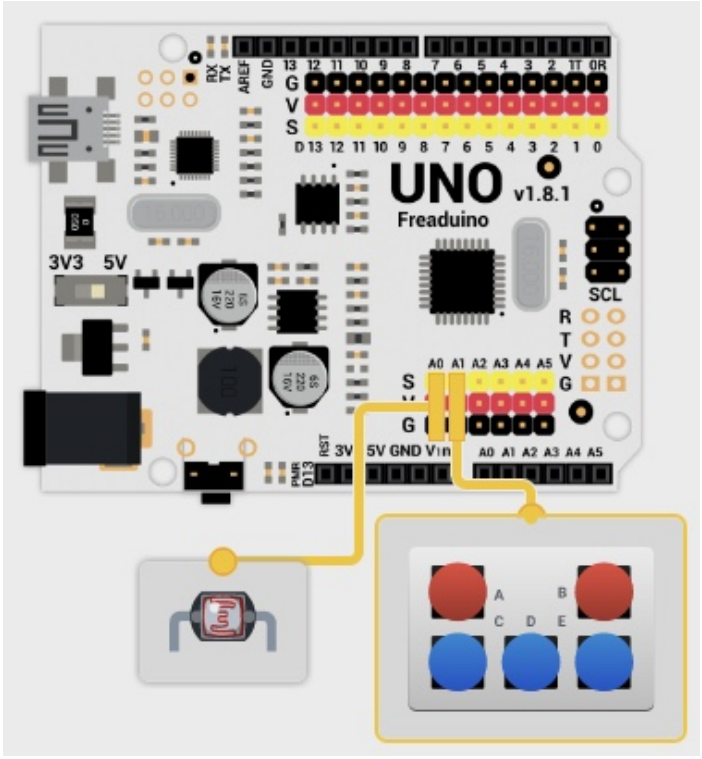

Figura 27. Resultado de conectar dos componentes analógicos

Conexión de componentes digitales y analógicos

Se realizará las conexiones entre la placa de desarrollo y varios componentes de tipo digital y analógico, los cuales se conectará a los pines adecuados. Las conexiones realizadas, así como los componentes y la placa elegida, quedan mostrada en la figura que aparece a continuación, figura 28. En esta ocasión se ha seleccionado como placa Arduino UNO, como componentes digitales el LED, el botón y el bluetooth, y como componentes analógicos la botonera y el sensor de temperatura y humedad. Añadir que este último componente tiene pines asignados por lo que no es necesario realizar la conexión, ya que se conecta automáticamente a estos pines asignados.

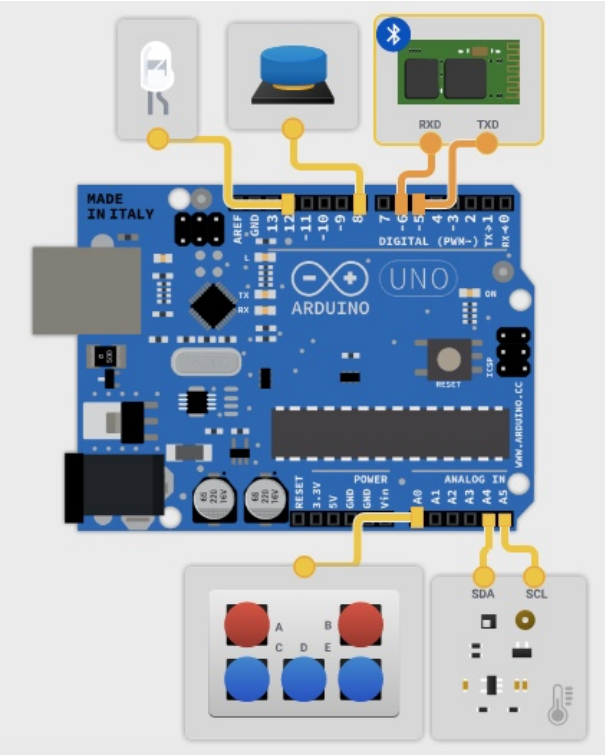

Figura 28. Conexión de componentes analógicos y digitales

Como resultados de esta prueba, obtendremos una reorganización de los pines digitales, de forma que se colocarán en pines consecutivos. En cuanto a los pines analógicos, no se obtendrá ninguna reorganización, ya que el pin de la placa que está utilizando la botonera coincide con el primer pin analógico disponible que devuelve el programa, y en cuanto al otro componente analógico, como hemos comentado anteriormente tiene pines asignados por lo que no se reorganizará sus conexiones. Los resultados comentados se pueden observar en la figura 29.

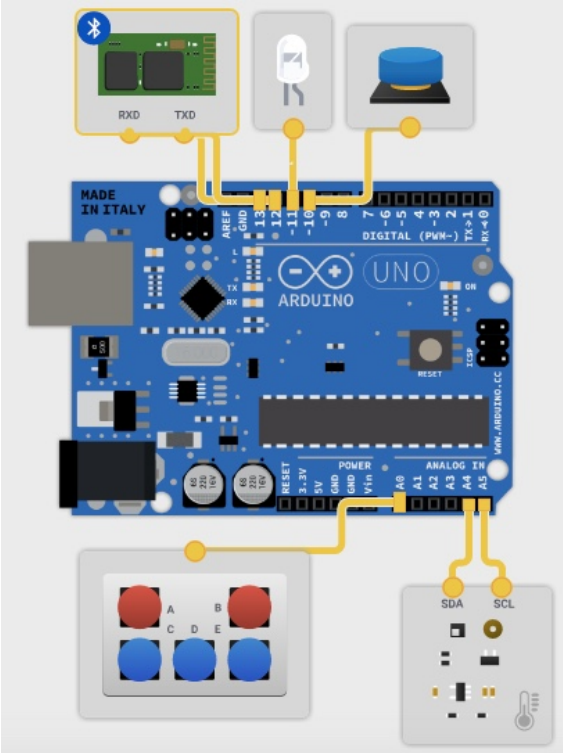

Figura 29. Resultado de conectar componentes analógicos y digitales

Conectar un componente sin pines asignados y otro componente con pines asignados al mismo pin

Esta prueba consiste en seleccionar un componente que no tenga pines asignados y realizar la conexión con la placa de desarrollo seleccionada. Posteriormente, se conectará un componente con pines asignados y que entre esos pines se encuentre el pin que hemos utilizado para el primer componente añadido. Las figuras 30 y 31, muestran estos pasos respectivamente.

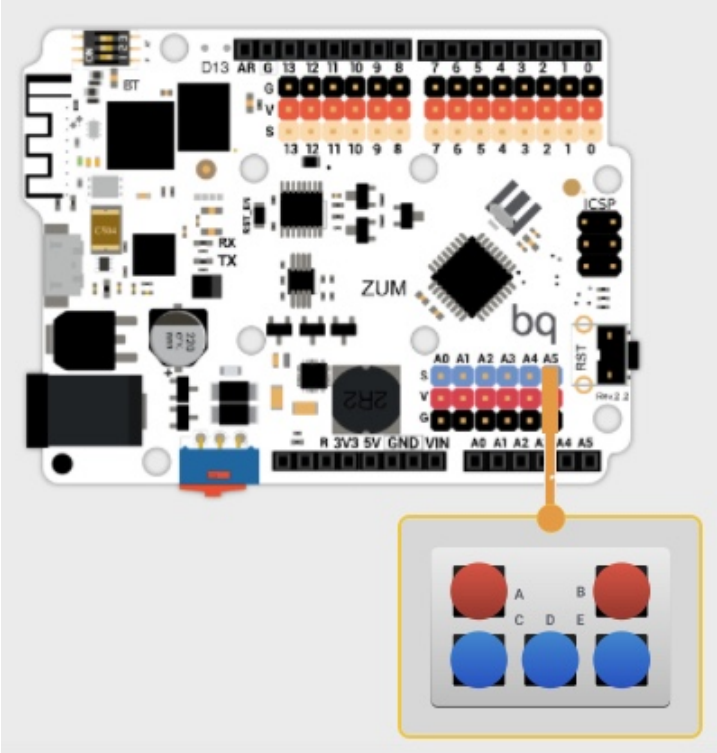

Figura 30. Conexión de componente analógico sin pines asignados

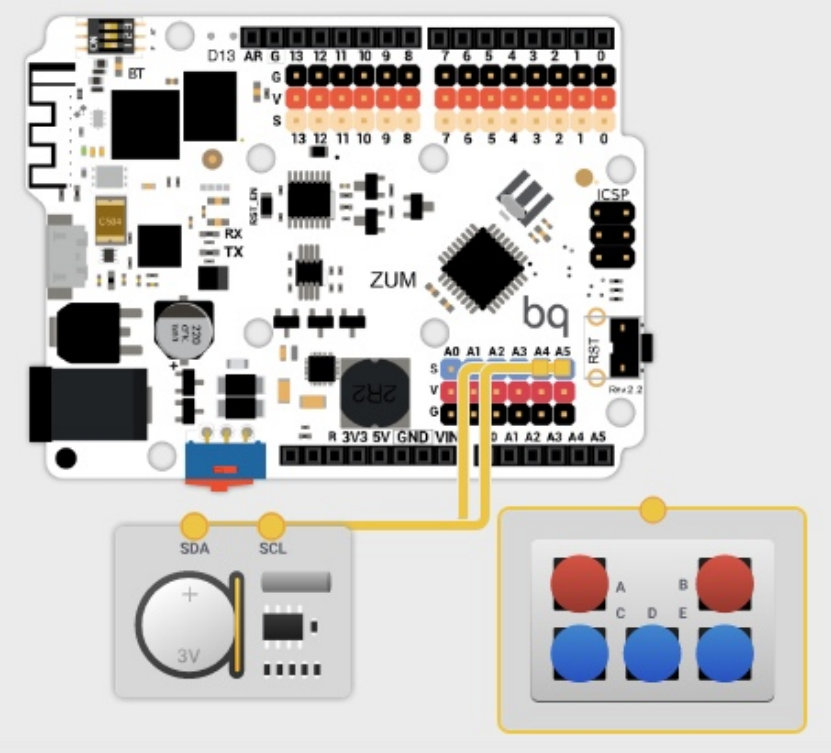

Figura 31. Conexión de componente analógico con pines asignados

Como se puede observar, el componente que no tiene pines asignados, en nuestro caso la botonera, se queda desconectado hasta que el usuario, de nuevo, realice una conexión con este componente.

Si realizamos estos mismos pasos en nuestro programa, la botonera recibirá una nueva conexión y no se quedará desconectada. Esto se debe a que nuestro programa chequea los componentes que han sido desconectados por estas circunstancias y realiza una nueva conexión siempre que queden pines libres de la placa y que sean del mismo tipo que el pin del componente. En las siguientes figuras, 32 y 33, se puede observar este resultado.

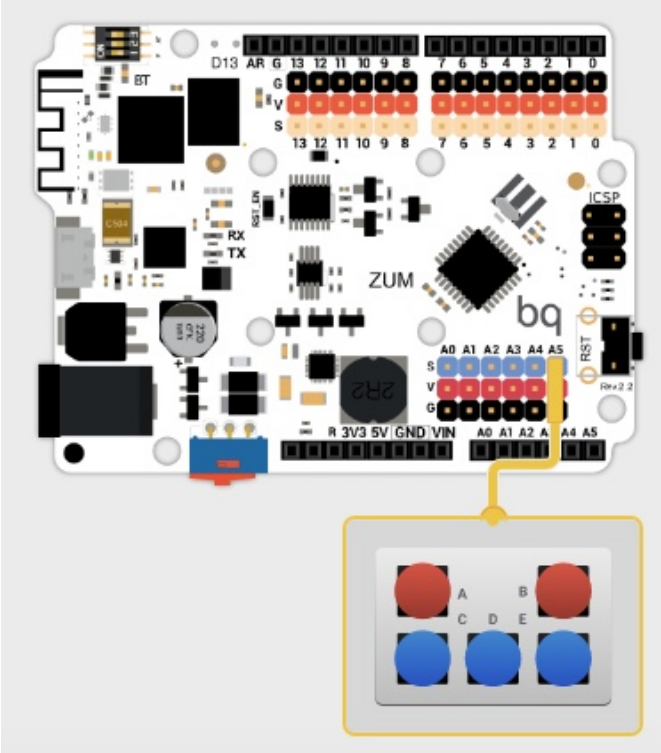

Figura 32. Conexión de componente analógico sin pines asignados

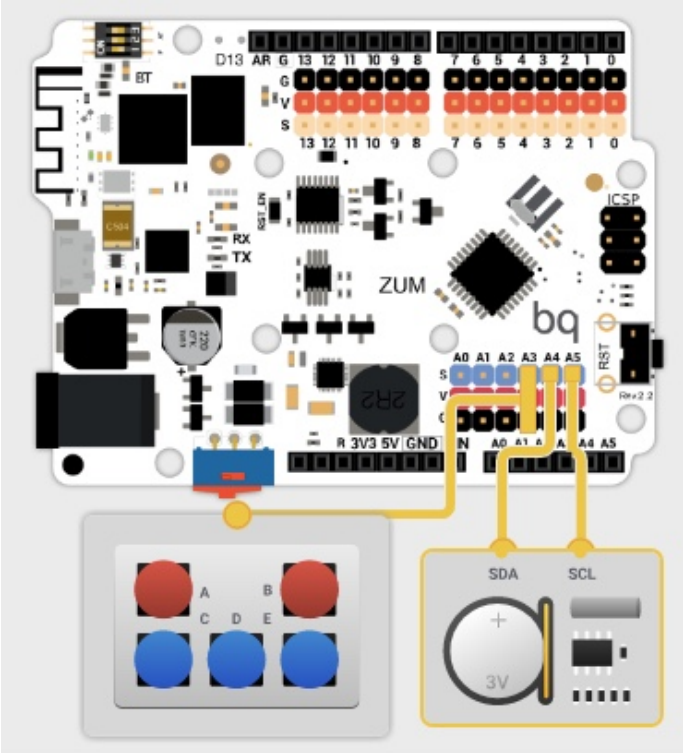

Figura 33. Conexión de componente analógico con pines asignados

Para realizar esta prueba se ha seleccionado la placa de desarrollo BQ ZUM, la botonera y el reloj de tiempo real como componentes analógicos. El primero corresponde al componente sin pines asignados y el segundo al componente con pines asignados.

Realizar las conexiones entre los componentes y una placa seleccionada y luego, cambiar la placa de desarrollo

Esta prueba realizada consiste en llevar a cabo las conexiones entre varios componentes y la placa de desarrollo. Cuando se encuentre todos los componentes conectado, se seleccionará otra placa distinta a la elegida. Esta prueba queda reflejada visualmente en las figuras 34 y 35, respectivamente.

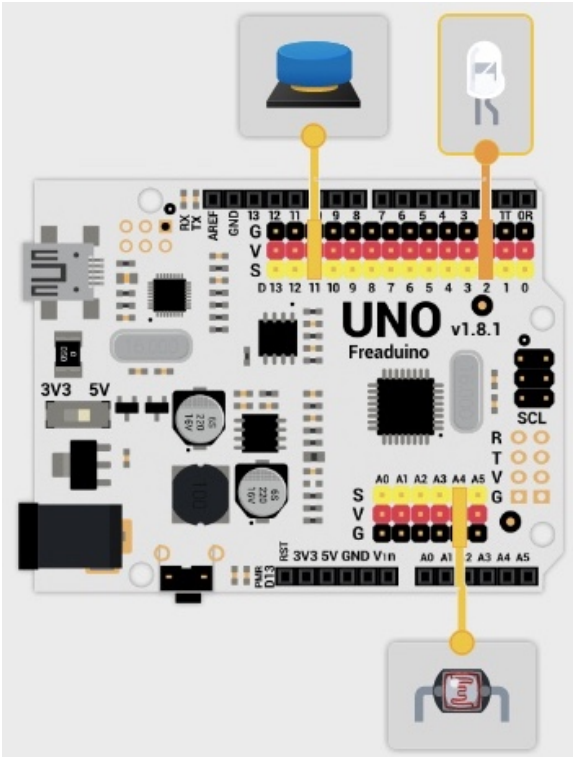

Figura 34. Conexión de componentes analógicos y digitales

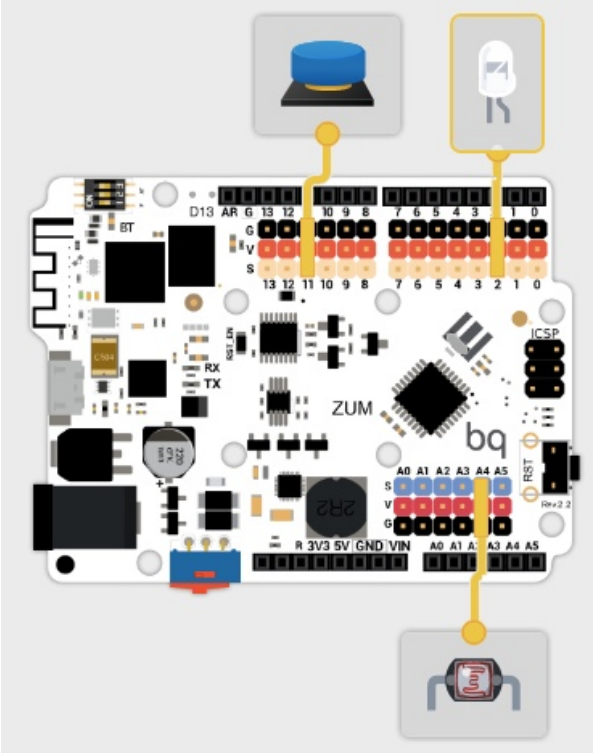

Figura 35. Conexión de componentes digitales y analógicos

En esta ocasión, los resultados de esta prueba serán dos, el primero, el resultado de reorganizar las conexiones con la primera placa elegida que corresponde con la Freaduino Uno y que se muestra en la figura 36.

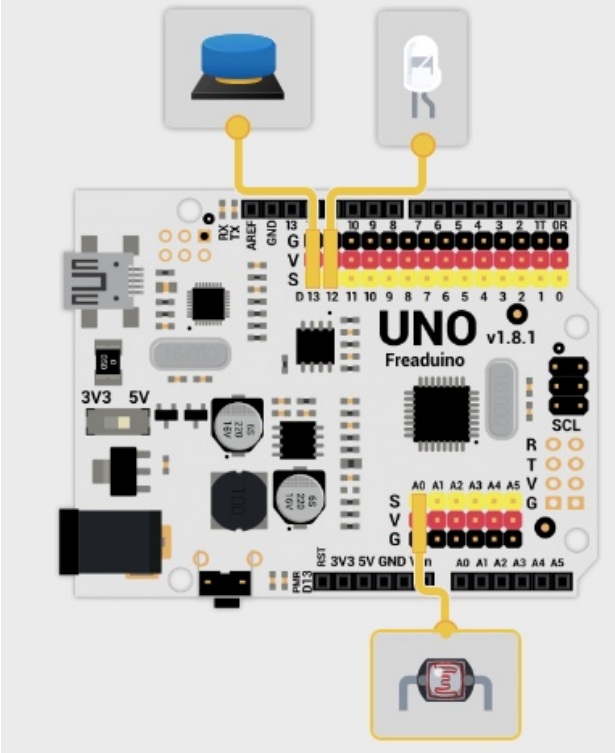

Figura 36. Conexión de componentes digitales y analógicos

El segundo resultado, la reorganización de las conexiones en la segundo placa seleccionada, que corresponde con la BQ ZUM. Este resultado se muestra a continuación, en la figura 37.

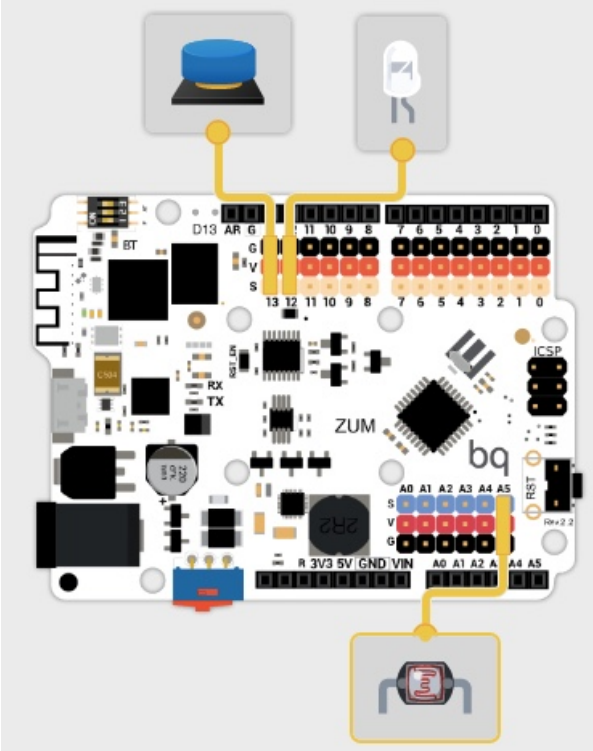

Figura 37. Conexión de componentes digitales y analógicos

Como podemos observar en las figuras anteriores, los resultados de esta prueba no coinciden entre sí; concretamente, nos referimos a la conexión del componente analógico tras realizar la reorganización de las conexiones. Esto se debe a que los pines analógicos de estas placas reciben identificadores diferentes y nuestro programa recorre los pines con un orden ascendente respecto a estos identificadores.

6. Conclusiones y líneas futuras

Como estudiante del Grado de Ingeniería de Computadores he adquirido algunos conocimientos tanto teóricos como prácticos que me han llevado a poder abarcar este trabajo fin de grado, lo cual no significa que no se haya encontrado dificultades en la elaboración del mismo.

En cuanto al cumplimiento de los objetivos iniciales del proyecto, hay que decir que no se han podido cumplir, ya que nos surgieron necesidades que debíamos solucionar antes de desarrollar los objetivos que se propuso inicialmente. Por lo que, estas necesidades se convirtieron, junto a otras extensiones realizadas, en el objetivo final del proyecto, las cuales se han cumplido satisfactoriamente.

Durante el desarrollo del proyecto, encontramos varias dificultades que tras un tiempo pudimos resolver. La primera de ellas fue el despliegue del entorno de desarrollo en el equipo local, al ser un entorno en el que hay que considerar varias partes, tuvimos problemas de configuración y de documentación, ya que la única ayuda que se proporciona en los distintos repositorios del proyecto en Github se encuentra en la parte del frontend. Estos problemas se pudieron solucionar contactando con la ayuda de algunos de los desarrolladores de BQ encargados del proyecto Bitbloq.
Otra de las dificultades que se encontraron fueron en cuanto a las tecnologías utilizadas en el proyecto, ya que nunca se había tenido contacto con estas y eran totalmente nuevas para nosotros. Por otro lado, nos resultó difícil entender la estructura del proyecto y la funcionalidad del código ya desarrollado, ya que nunca es fácil entender código de otros desarrolladores, y más cuando se trata de un proyecto de estas dimensiones, como es Bitbloq.

En general, el grado de satisfacción es alto por haber conseguido finalizar el proyecto y poder cumplir con los objetivos que se fijaron. Este proyecto nos ha ofrecido la oportunidad de adquirir conocimientos en estas tecnologías, no se descarta seguir desarrollando con dichas tecnologías, y de poder aportar nuestro grano de arena en la acogida de la programación en la educación.

Para finalizar este apartado, propondremos algunas líneas futuras para este proyecto.

Una posible e interesante línea de trabajo consistiría en implementar sobre nuestro proyecto una capa de abstracción del mismo nivel de la que ofrece Scratch, de forma que, por ejemplo, las placas de desarrollo se vean como una estancia de una casa, y los componentes, como un termómetro, luces, etc. De esta forma, el usuario no tendría por qué saber el hardware que se está utilizando a bajo nivel.

Esta implementación necesitaría de otra capa software que se encargue de hacer la elección de la placa de desarrollo y de los componentes necesarios para hacer posible la conexión entre lo desarrollado en este proyecto y esta nueva capa de abstracción. Esta parte también se propone como línea futura a implementar.

También se podría exponer como línea futura, implementar lo desarrollado en este proyecto a la versión offline de Bitbloq. Esta versión es más simple que la online y se desarrolló para poder utilizar la plataforma sin tener que hacer uso de la red.

Apéndice I Manual de Usuario

En esta sección del documento vamos a realizar una explicación sobre cómo se llevaría a cabo el despliegue del entorno de ejecución del proyecto sobre el sistema operativo MacOS, concretamente para la versión 10.12.4, que es la que se ha utilizado para el desarrollo.

Antes de comenzar, recordar que el proyecto se ha realizado sobre el proyecto de código abierto Bitbloq. Este proyecto está dividido en varias partes, las más importante son:

- Backend: enlace [10] de la bibliografía
- Frontend: enlace [11] de la bibliografía
- Bloques: enlace [12] de la bibliografía

En primer lugar, comenzaremos con la parte del backend. Esta parte ofrece una interfaz de programación basada en REST (Transferencia de Estado Representacional), es decir, una interfaz que utiliza el protocolo HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON, y necesita un servicio de bases de datos; concretamente, se utiliza MongoDB, como se comentó en el apartado 3 de este documento.

Para la instalación de MongoDB se puede llevar a cabo los diferentes pasos que aparecen en la web que se referencia en el enlace [13] de la bibliografía.

Una vez realizada la instalación, se necesitará realizar una copia local del repositorio, es decir, se necesita clonar el repositorio que se encuentra en Github. Para ello se va a hacer uso del servicio de Git, en el caso de que dicho servicio no se encuentre instalado en el sistema, se podrá seguir los pasos que aparece en la documentación oficial de Git, a la cual se hace referencia en la bibliografía, concretamente el enlace [14].

Una vez instalado, procederemos a ejecutar el clon del repositorio, para ello utilizamos el siguiente comando: *git clone https://github.com/<usuario_github>/bitbloq-backend.git*. Si se quieren realizar modificaciones en el código, antes de ejecutar el comando anterior, se deberá generar una rama a una cuenta propia de Github. De esta forma, se podrán subir los cambios que se hagan; si solo se clona, no se pueden subir cambios al repositorio. Se puede encontrar más información sobre las bifurcaciones en Github en el enlace [15] de la bibliografía. Una vez que se tenga el repositorio en la cuenta de Github sí se procederá a realizar el clonado. En nuestro caso, esta es la acción que se ha necesitado realizar.

Se necesitará de la instalación de NodeJS para la ejecución del backend. Para ello, se podrá descargar el instalador desde la página oficial, enlace [16], de NodeJS y ejecutarlo.

Una vez instalado, nos dirigimos al directorio local donde hemos clonado el repositorio, en nuestro caso, el directorio se corresponde a */Users/<nombre_usuario>/Desarrollo/Bitbloq/bitbloq-backend* y ejecutamos el comando *npm install*, este comando instalará todos los paquetes software de NodeJS necesarios para el funcionamiento del backend.

Antes de ejecutar el programa, necesitaremos añadir un fichero de configuración; dicho fichero se almacenará en *app/res/config/config.json* y tendrá un contenido como el que se muestra en la figura 38:

```
{
    "env": "local",
    "gcloud": {
        "projectId": " ",
        "keyFilename": ""
    },
    "cloudStorageBucket": " ",
    "client_domain": "http://localhost:9000",
    "secrets": {
        "session": " "
    },
    "port": 8000,
    "ip": "127.0.0.1",
    "mongo": {
        "uri": "mongodb://localhost/bitbloq"
    },
    "seedDB": false,
    "mailer": {
        "auth": {
            "user": " ",
            "pass": " "
        },
        "defaultFromAddress": " "
    },
    "supportEmail": "",
    "userRoles": [
        "guest",
        "user",
        "admin"
    ]
}
```


Figura 38. Fichero configuración backend

Los campos más importantes son:

- mongo: Donde se indicará la dirección a la base de datos a la cual accederá el backend. En nuestro caso la hemos nombrado “bitbloq”.
- env: Que indicará si se trata de un entorno de desarrollo o de producción; para indicar que es de desarrollo asigna el valor “local”.
- client_domain: En este campo corresponde con la dirección y puerto donde se está ejecutando el frontend.
- port: Indica el puerto en el que atenderá las peticiones el backend.
- ip: La dirección ip del backend.
- userRoles: Indica los distintos roles que los usuarios registrados pueden tener.

Estos son los campos que mínimamente hay que proporcionar para que el backend se ejecute correctamente. Los demás datos no son necesarios, ya que es información para otro tipo de servicio, los cuales son compatibles con el backend.

Una vez realizado los pasos anteriores, desde la consola, nos ubicamos en el directorio del proyecto y ejecutamos los siguientes comandos:

- *mongod –dbpath <dirección a directorio de datos>* si hemos cambiado la localización por defecto del directorio de datos; si no es así, nos valdría con el comando *mongod*
- *npm start*

El primero iniciará el servicio de MongoDB y el segundo lanzará el backend, el cual se quedará esperando peticiones. Para probar su funcionamiento, se puede escribir en el navegador, por ejemplo, la siguiente dirección *http://localhost:8000/bitbloq/v1/version*

Una vez finalizados los pasos para el backend, se continuará con los pasos para el frontend. En este caso debemos realizar la misma tarea que realizamos con el repositorio del backend, es decir, clonarlo al entorno local, si no se va a modificar el código, o realizar una rama a una cuenta propia de Github y luego realizar un clon, en el caso de que se quiera modificar el código del frontend, ya que es la única forma de hacerlo si se quiere subir los cambios realizados.

Para realizar el clon del repositorio, utilizamos el siguiente comando desde la consola del equipo: *git clone https://github.com/<usuario_github>/bitbloq-frontend.git.*

Una vez que hemos realizado el clon del repositorio, nos dirigimos al directorio, en nuestro caso, el directorio se corresponde a */Users/<nombre_usuario>/Desarrollo/Bitbloq/bitbloq-fronted* y ejecutamos el comando *npm install*. Este comando instalará todos los paquetes de NodeJS necesarios para el funcionamiento del frontend. Esta vez, también necesitaremos ejecutar el comando *bower install,* que instalará los paquetes software necesarios para el navegador, es decir, paquetes de HTML, CSS…

En caso de no tener instalado el gestor de paquete Bower, se podrá instalar a través de la consola del equipo ejecutando el comando *npm install –g bower*, información referenciada en el enlace [17] de la bibliografía.

Antes de poner en marcha el frontend, necesitaremos crear tres ficheros de configuración. Estos ficheros se almacenarán en la siguiente ruta del proyecto: *app/res/config.* El contenido de los ficheros se podrá encontrar en el repositorio, al cual se hace referencia en el enlace [11] de la bibliografía. Todos los campos de estos ficheros se pueden dejar sin modificar, pero añadir, que el campo más importante es "serverURL", correspondiente al fichero *config.json*, el cual indica la dirección y el puerto donde se está ejecutando el backend. Además de estos ficheros, existe un fichero llamado *Gruntfile.js*, en la raíz del proyecto, que contiene algunas configuraciones más del servidor, como, por ejemplo, el puerto en el que atiende las peticiones, que, por defecto, está configurado como el puerto 9000.

Una vez terminado los pasos anteriores, ejecutaremos desde el directorio del proyecto y a través de la consola el comando *grunt server,* al ejecutar este comando, se realiza la compilación del proyecto (minimiza ficheros, genera CSS a partir de un SASS...), lanza un servidor en local con la web y lanza un proceso que refresca automáticamente el navegador cuando detecta cambios en algunos de los archivos del proyecto.

Si no se tiene instalado el gestor de tareas Grunt para NodeJS, se podrá instalar a través de la ejecución del comando *npm install –g grunt-cli.* La documentación de este gestor, así como el método para su instalación se puede encontrar en la página web que referencia el enlace [18] de la bibliografía.

El último paso para tener por completo el entorno en local, es añadir los bloques a la base de datos. Para ello, se necesitará registrar un usuario y hacerlo administrador del sistema, ya que es una acción que se realiza a través de la API REST y que solo es permitida a los usuarios con estas características.

Para poner a un usuario como administrador, hay que modificarlo desde la base de datos. Se modificará cualquier usuario creado y se le cambiará el campo role “user” por “admin”, con estas modificaciones, ya se podrá realizar la acción.

Respecto a los bloques, están en una carpeta llamada *dataBaseFiles* dentro del proyecto del frontend, para subirlos al servidor, se ejecutará una tarea de Grunt que se encarga de hacer las llamadas necesarias de la API.

Desde la terminal del equipo y situado en la localización del proyecto, se tendrá que ejecutar el comando *grunt updateAllCollections*, dicho comando actualiza la base de datos con el contenido de los ficheros de dicha carpeta.

Para no tener que autenticarse cada vez que se tenga que realizar una petición al backend, existe un fichero en la raíz del proyecto que se llama *gruntconfig.json* con estos campos:

```
{
  "adminUser": "email_usuario_bitbloq",
  "adminPassword": "contraseña_usuario_bitbloq"
}
```

Figura 39. Email y contraseña usuario administrador

Una vez que se crea el fichero con los campos del usuario que se ha configurado como administrador, el comando anterior se ejecutará con éxito.

Después de realizar estos pasos, habrá que garantizar que la configuración de las direcciones de los servidores y de los puertos están correctamente, es decir, asegurarnos de que el fichero de configuración del backend contiene la dirección y puerto del frontend correctamente y viceversa.

Apéndice II Manual de Instalación

En este apéndice se explicará los pasos que habrá que seguir para poder hacer uso del código desarrollado en el proyecto.

Como se puede observar, en el soporte electrónico entregado con este proyecto, encontraremos dos ficheros con extensión de JavaScript; estos dos ficheros reemplazarán a los dos ficheros correspondientes originales del proyecto.

Comenzaremos por el fichero que tiene como nombre *hardwareTab.js*. Dicho fichero debemos copiarlo en la siguiente ruta del proyecto: */bitbloq-fronted/app/scripts/controllers*. Este fichero, como se indicó en el apartado 4 de este documento, contiene las funciones necesarias para hacer la lectura de los ficheros JSON con el contenido de los componentes o de las placas de desarrollo; más adelante se explicará en qué directorio habrá que crear esos ficheros.

El siguiente fichero se corresponde con *hw2bloqs.js*, que habrá que copiarlo en el directorio */bitbloq-fronted/app/scripts/services*. En este fichero, se encuentra el algoritmo que ha desarrollado para la reordenación de las conexiones.

Una vez copiado ambos ficheros, crearemos los directorios correspondientes para almacenar los ficheros JSON con la información de las placas y de los componentes. Para los componentes, se creará el directorio *components* en la ruta del proyecto */app/json*. En cuanto a las placas habrá que proceder del mismo modo, pero creando el directorio *boards.* Originalmente el directorio *json* no se localizará en el proyecto, por lo que también necesitaremos hacer la creación de este directorio.

Realizados estos pasos, se deberá ejecutar, en el caso de que no se esté ejecutando ya el sistema, el comando *grunt serve* desde la consola del equipo y situados en la ruta donde se tenga almacenado el proyecto del frontend. Este comando, como se explicó en el anterior apéndice, se encarga de compilar el proyecto, por lo que detectará los cambios que hemos realizado y se ejecutará el nuevo código. También

habrá que proceder a ejecutar el backend e iniciar el servicio de base de datos para que el sistema funcione de forma completa.

Bibliografía

[1] La programación informática como herramienta didáctica. Disponible en: http://blog.educalab.es/intef/2014/02/26/la-programacion-informatica-como-herramienta-didactica/

[2] Cómo iniciar a un niño en la programación desde cero. Disponible en: https://www.xataka.com/otros/como-iniciar-a-un-nino-en-la-programacion-desde-cero

[3] Desarrollo iterativo y creciente. Disponible en: https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[4] Documentación sobre objetos JavaScript de tipo Blob. Disponible en: https://developer.mozilla.org/es/docs/Web/API/Blob

[5] Documentación sobre objetos JavaScript de tipo FileReader. Disponible en: https://developer.mozilla.org/es/docs/Web/API/FileReader

[6] Creación de cadena JSON a partir de objetos JavaScript. Disponible en: https://www.w3schools.com/js/js_json_stringify.asp

[7] Definición y utilización del localStorage en html5. Disponible en: https://rolandocaldas.com/html5/localstorage-en-html5

[8] Documentación sobre el servicio de http en AngularJS. Disponible en: https://docs.angularjs.org/api/ng/service/$http

[9] Información sobre los objetos Popup Boxes de JavaScript. Disponible en: https://www.w3schools.com/js/js_popup.asp

[10] Repositorio de Github del backend de Bitbloq. Disponible en: https://github.com/bq/bitbloq-backend

[11] Repositorio de Github del fronted de Bitbloq. Disponible en: https://github.com/bq/bitbloq-frontend

[12] Repositorio de Github de los bloques de Bitbloq. Disponible en: https://github.com/bq/bloqs

[13] Instalación de MongoDB en OS X 10.7 y superior. Disponible en: https://docs.mongodb.com/master/tutorial/install-mongodb-on-os-x/?_ga=2.198813990.84064399.1497450766-1238155588.1486583928

[14] Instalación de Git. Disponible en: https://git-scm.com/book/es/v1/Empezando-Instalando-Git

[15] Información sobre bifurcación en Github. Disponible en: https://help.github.com/articles/fork-a-repo/

[16] Página web de NodeJS. Disponible en: https://nodejs.org/es/

[17] Instalación del gestor de paquetes Bower. Disponible en: https://bower.io/

[18] Instalación del gestor de paquetes Grunt. Disponible en: https://gruntjs.com/getting-started

[19] Documentación de la función sort() de JavaScript. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/sort